Alpiflow

The Five-Day Database Audit

Sample report on a synthetic pgbench workload · public extract

# Sample Co (synthetic)

PostgreSQL 16.15 · database postgres

| | |
|---|---|
| Client | Sample Co (synthetic) |
| Report date | 2026-11-01 |
| Bundle collected | 2026-11-01 11:14 UTC |
| Report version | 1.0 |
| Prepared for | Head of Engineering (synthetic) |

Nothing in this report comes from a client. How the database was built is on the last page, “About this sample”.

Prepared by Alpiflow, independent engineering studio · hello@alpiflow.com

# Summary

Sample Co runs PostgreSQL 16.15 with one 580 MB application database and four tables (version.csv, databases.csv). The counters cover 28 minutes that include a 10-minute load (version.csv, uptime 00:28:23). Half of the workload's execution time goes to one lookup on `pgbench_history`, a table with no index: 61,843 calls, 510,048 ms (statements-by-total-time.csv, row 1). The busiest write, the account update, maintains three indexes where one would do, and one of them blocks HOT updates: 53 of 561,996 updates were HOT (tables.csv). We found no data risk: transaction ID age is 562,117 against an `autovacuum_freeze_max_age` of 200,000,000, and no session was blocked at collection time.

## The interventions, ranked

Ranked by risk removed and expected benefit against effort. Each row has its own chapter and script.

| # | Intervention | Risk of change | Effort | Expected benefit | Script |
|---|---|---|---|---|---|
| 1 | Index the lookup on `pgbench_history (aid)` | ☐ Low | 15 min | The lookup reads a few index pages instead of the whole table | `01-index-pgbench-history-aid.sql` |
| 2 | Drop the duplicate index `pgbench_accounts_aid_dup` | ☐ Low | 15 min | One index less to write on every account update; 64 MB freed | `02-drop-duplicate-accounts-index.sql` |
| 3 | Drop the unused index `pgbench_accounts_abalance` after one business cycle | ◪ Medium | 15 min, after a wait | Account updates can be HOT; 24 MB freed | `03-drop-unused-abalance-index.sql` |
| 4 | Vacuum `pgbench_accounts` sooner | ◪ Medium | 30 min | Dead rows cleaned at 150,059 instead of 600,085 | `04-autovacuum-pgbench-accounts.sql` |
| 5 | Raise `max_wal_size` to 4 GB | ☐ Low | 15 min | Checkpoints follow the 5-minute timer, not WAL volume | `05-max-wal-size.sql` |
| 6 | Turn autovacuum back on for `pgbench_tellers` | ☐ Low | 15 min | Statistics and cleanup resume on a table updated 561,996 times | `06-reenable-autovacuum-pgbench-tellers.sql` |
| 7 | Turn on I/O timing, slow statement and lock wait logging | ☐ Low | 15 min | Disk reads, lock waits and slow statements become visible | `07-monitoring-settings.sql` |

## Measured on the sample instance

This database is synthetic and disposable, so we also applied the scripts to it and measured. A client audit never includes this step: we do not access client systems.

| Metric | Before | After | Source |
|---|---|---|---|
| Lookup on `pgbench_history`, latency per call | 19.443 ms | 0.246 ms (script 01) | pgbench, runs A2 and B |
| Blocks read per lookup | 3,732 | 3.2 (script 01) | pg_stat_statements |
| Account updates done as HOT | 1 of 57,716 | 62,320 of 62,431 (scripts 02 and 03) | pg_stat_user_tables, runs B and C |
| WAL per account update | 6,528 bytes | 2,866 bytes (scripts 02 and 03) | pg_stat_statements, runs B and C |
| Indexes on `pgbench_accounts` | 153 MB | 64 MB (scripts 02 and 03) | pg_indexes_size |
| Throughput | 847.9 tps | 1,158.7 tps (01 to 03); 1,223.2 tps (all seven) | pgbench, runs A2, C, D |

One 60-second run per step, on a workstation shared with other services. These figures show the direction of each change on this data. They are not a forecast for another server. Run details are in the full sample.

# 1. Index the lookup on pgbench_history (aid)

☐ **Risk of change:** Low `Effort: 15 min` `Script: scripts/01-index-pgbench-history-aid.sql`

## Evidence

| Object | Metric | Value | Source |
|---|---|---|---|
| `SELECT count(*), sum(delta) FROM pgbench_history WHERE aid = $1` | calls / total time | 61,843 / 510,048.12 ms | statements-by-total-time.csv, row 1 |
| same | share of all recorded time / of workload time | 46.96% / 50.60% | same row, `pct_total_time`; workload total above |
| same | mean / max per call | 8.247 ms / 39.003 ms | same row |
| same | shared blocks hit | 111,217,394 (1,798 per call) | same row |
| `pgbench_history` | seq_scan / seq_tup_read | 109,901 / 17,415,312,242 | tables.csv |
| `pgbench_history` | est_rows / size / index bytes | 557,812 / 28 MB / 0 | tables.csv |
| `pgbench_history` | rows inserted in the window | 561,996 | tables.csv, `n_tup_ins` |

## Why it matters

There is no index on `pgbench_history`, so every lookup reads every row to find the one or two it needs. The table grows by one row per business transaction, so each lookup gets slower as the day goes on. The `seq_scan` count is higher than the number of calls because parallel workers count their own scans. On the sample instance the plan was a Parallel Seq Scan with two workers. This single statement is half of the workload's execution time.

## What to do

```
-- Core of scripts/01-index-pgbench-history-aid.sql (psql, outside a transaction block)
SET lock_timeout = '5s';
SET statement_timeout = 0;
CREATE INDEX CONCURRENTLY IF NOT EXISTS pgbench_history_aid_idx
    ON public.pgbench_history USING btree (aid);
```

```
-- Rollback
DROP INDEX CONCURRENTLY IF EXISTS public.pgbench_history_aid_idx;
```

Measured on the sample instance (same lookup value, 608,244 rows in the table): 3,878 buffers and 13.8 ms with a sequential scan, 4 buffers and 0.034 ms with the index. Under load, the lookup's latency per call fell from 19.443 ms to 0.246 ms. The index is 13 MB and took 0.38 s to build.

## How to verify

```
SELECT seq_scan, seq_tup_read, idx_scan
FROM pg_stat_user_tables WHERE relid = 'public.pgbench_history'::regclass;

SELECT calls, total_exec_time, shared_blks_hit + shared_blks_read AS blocks
FROM pg_stat_statements
WHERE query LIKE 'SELECT count(*), sum(delta) FROM pgbench_history%';
```

Expected: `seq_scan` stops growing and `idx_scan` grows. Between two readings, blocks divided by calls drops to single digits (3.2 on the sample instance).

## Risks of this change

The build waits for transactions already running, then works alongside reads and writes. If it is interrupted it leaves an invalid index; the script detects that and rebuilds. Each insert into `pgbench_history` now also writes one index entry. On the sample instance the business transaction latency did not rise: 8.311 ms before, 8.258 ms after.

# Chapters 2 to 7

The full sample has a chapter for each intervention in the ranked table. Each one follows the structure of chapter 1: the evidence from the bundle, why it matters, the fix with its rollback, how to verify it and the risks of the change.

| # | Chapter | Risk of change | Script |
|---|---|---|---|
| 2 | Drop the duplicate index `pgbench_accounts_aid_dup` | ☐ Low | `02-drop-duplicate-accounts-index.sql` |
| 3 | Drop the unused index `pgbench_accounts_abalance` | ◩ Medium | `03-drop-unused-abalance-index.sql` |
| 4 | Vacuum `pgbench_accounts` sooner | ◩ Medium | `04-autovacuum-pgbench-accounts.sql` |
| 5 | Raise `max_wal_size` to 4 GB | ☐ Low | `05-max-wal-size.sql` |
| 6 | Turn autovacuum back on for `pgbench_tellers` | ☐ Low | `06-reenable-autovacuum-pgbench-tellers.sql` |
| 7 | Turn on I/O timing, slow statement and lock wait logging | ☐ Low | `07-monitoring-settings.sql` |

The full sample also contains what we looked at and found healthy, the observations that need an application change, the recommended configuration and an appendix with the data used. Ask for it at alpiflow.com/#contact.

# 90-day plan

The same plan, with owners and checks, is in `piano-90.md`.

## Week 1 (2026-11-02 to 2026-11-06)

| Day | Action | Script | Check |
|---|---|---|---|
| 2026-11-02 | Take the “before” readings of every chapter. Enable monitoring settings | 07 | `SHOW track_io_timing` in a new session |
| 2026-11-03 | Index the history lookup | 01 | `seq_scan` on `pgbench_history` stops growing |
| 2026-11-04 | Drop the duplicate index | 02 | `pgbench_accounts_pkey` gains scans |
| 2026-11-05 | Raise `max_wal_size` after checking WAL disk space. Re-enable autovacuum on tellers | 05, 06 | `SHOW max_wal_size`; tellers reloptions |
| 2026-11-06 | Second readings | none | Compare with 2026-11-02 |

## Month 1 (to 2026-12-01)

| Week of | Action | Script | Check |
|---|---|---|---|
| 2026-11-09 | Lower the autovacuum trigger on accounts | 04 | `autovacuum_count` grows |
| 2026-11-16 | Read `checkpoints_req` vs `checkpoints_timed` for the week | none | Requested checkpoints rare |
| 2026-11-23 | Decide on `shared_buffers` with the host’s memory figures; book a restart window | config | Host free memory |
| 2026-11-30 | Read `idx_scan` on `pgbench_accounts_abalance` after the November month-end | none | Still 3 |

## Month 3 (to 2027-02-01)

| Action | Why it waits | Check |
|---|---|---|
| Drop `pgbench_accounts_abalance` (script 03), from 2026-12-01 | The statistics window was 28 minutes | HOT share of account updates |
| Restart with the new `shared_buffers`, if approved | Needs a restart | Heap cache hit on `pgbench_accounts` |
| Primary key on `pgbench_history` (O2) | Exclusive lock, insert statements to check | Table gone from constraints-no-pk.csv |
| Application role without superuser (O5) | Needs an app configuration change | roles.csv |
| Branch totals redesign (O1) | Needs an app release | Branch update share of workload time |

# About this sample

This report describes a synthetic database, built to show what The Five-Day Database Audit delivers. Sample Co does not exist and nothing here comes from a client.

How the database was built:

- `pgbench -i -s 30` on PostgreSQL 16.15: 3,000,000 rows in `pgbench_accounts`, `pg_stat_statements` enabled.
- Planted before the load: a duplicate index `pgbench_accounts_aid_dup` on `pgbench_accounts (aid)`, an unused index `pgbench_accounts_abalance`, and autovacuum switched off on `pgbench_tellers`. `pgbench_history` has no primary key by pgbench's own design.
- Load: 10 minutes of `pgbench -c 8 -j 2 -T 600`, mixing 90% of the built-in TPC-B-like script with 10% of a custom lookup, `SELECT count(*), sum(delta) FROM pgbench_history WHERE aid = :aid`.
- At the end of the load we ran our read-only collection script as a superuser on database `postgres` and analysed the bundle exactly as we would for a client.

Every finding rests on the bundle's numbers, with file and value. The planted problems surfaced from the data (chapters 2, 3 and 6 and observation O2 of the full sample). The lookup was added to the load on purpose, and pgbench creates no index for it (chapter 1). Other findings were not planted: the autovacuum trigger on the largest table, checkpoints driven by WAL volume, the monitoring gaps and the hot branch row.

Because this instance was disposable, we also applied the scripts to it and measured before and after. That step is not part of a client audit: you run our script, you send us the bundle, and we never touch your systems.

---

Questions on this report are included for two weeks after delivery: hello@alpiflow.com